

Creating Fuzzy Micros

The foundations for fuzzy logic were laid by Professor Lotfi A. Zadeh of U.C. Berkeley in 1965 but, until recently, the field has been little more than a curiosity to engineers in the U.S. American attitudes began to change when an explosion of practical and very successful applications began to appear in Japan. The fuzzy revolution in Japan has been so strong that "fuzzy" has become a

household word that is universally understood to mean smart, as in having the ability to think like a human. Now, American engineers are scrambling to catch up. The good news is that fuzzy logic is a very accessible technology because it is largely based on common sense at the application level.

This article differs from most available literature on this subject because it is written for the embedded systems programmer and explains detailed al-

gorithms for implementing fuzzy-logic inference on a conventional microcontroller. While many consumer and industrial control applications can immediately use software implementations of fuzzy logic, dedicated fuzzy processors will be required for some high-performance applications.

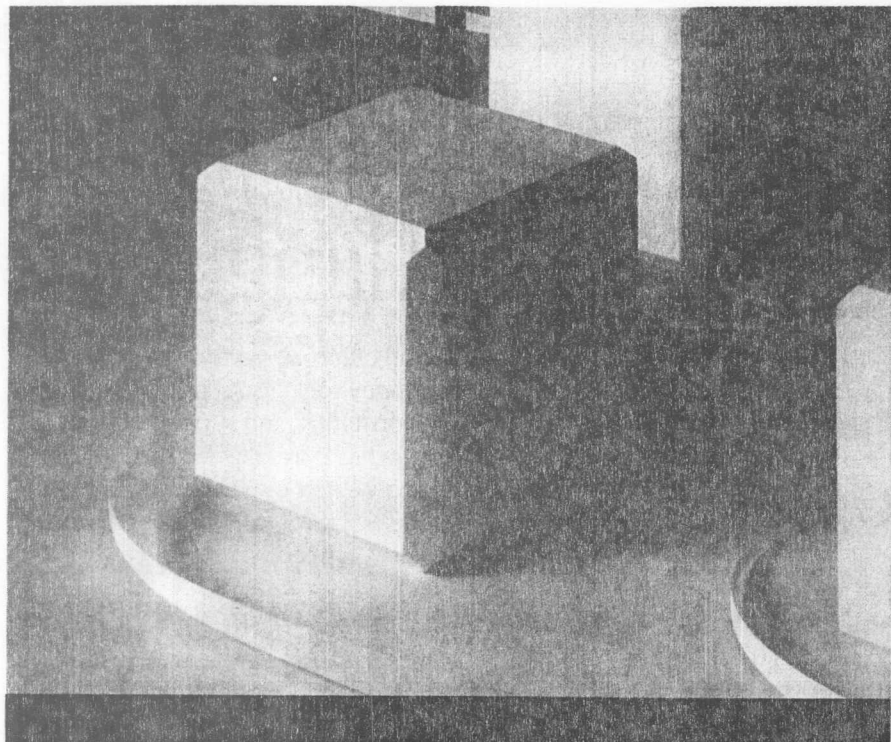
ARISTOTLE VS. FUZZY LOGIC

In western cultures, formal logic is rooted in the teachings of Aristotle. This framework is based on the idea that a specific piece of data is or is not a member of a particular set. This bivalent logic is very convenient and easy to work with, especially for digital computers but, unfortunately, it does not describe most natural systems very well.

Fuzzy logic is a more general logic system that allows for formal manipulation of sets whose boundaries are fuzzy. An input data point may be partially a member of a set and may be partially a member of the complementary set at the same time. To someone grounded in Aristotelian logic, this contradiction appears intractable, yet these situations are commonplace in the natural world. A glass does not suddenly make the transition from empty to full. Empty and full are fuzzy concepts that can be described by fuzzy membership functions.

Consider the following example of a merry-go-round. A mother tells a child not to get on or off the merry-go-round until it has stopped. This rule is obviously intended to prevent injury to the child. Aristotle's logic works pretty well until Mom wants to go home. The child says, "O.K., we'll go home as soon as the merry-go-round stops." When the merry-go-round slows to a few RPM, Mom says, "That's close enough, let's go."

Aristotle's logic is inadequate to explain what the term "stopped" means. A typical engineer might next try to establish some slow rate of rotation and say that anything less is equivalent to stopped and anything more is not stopped. In many cases, this method is good enough and, in fact, many control problems are currently being solved with similar compromises. As the com-



David Bishop

Fuzzy logic is a very accessible technology because it is largely based on common sense at the application level.

plexity of a control problem increases, this technique breaks down because the cut-off points do not provide a realistic representation of the meaning of the original information. Later, we will see how fuzzy logic allows seemingly contradictory rules to be combined to arrive at an appropriate control action.

Figure 1 shows the membership function for "stopped." The X axis of a membership function is called the universe of discourse and shows the range of possible values for an input variable. The Y axis represents the degree to which the corresponding input values are members of a set, where a value of zero indicates no membership and a value of one represents full membership. Y values between zero and one represent partial membership. For the merry-go-round example, the X axis might range from 0 RPM to 32 RPM. The meaning of "not stopped" is shown by the dashed membership function, which is found by taking one minus the membership function value of stopped for all values of X. When the speed is 0 RPM, the merry-go-round is completely stopped, at 2 RPM you would say it is stopped with a grade of membership of 0.5, and at 4 RPM you would say it is stopped with a grade of membership of zero. This method more realistically conveys the mother's meaning of the term "stopped" including the gray area between 0 RPM and 4 RPM, where the merry-go-round is simultaneously stopped to

some degree and not stopped at the same time. In Aristotle's bivalent logic, stopped and not stopped must be mutually exclusive conditions.

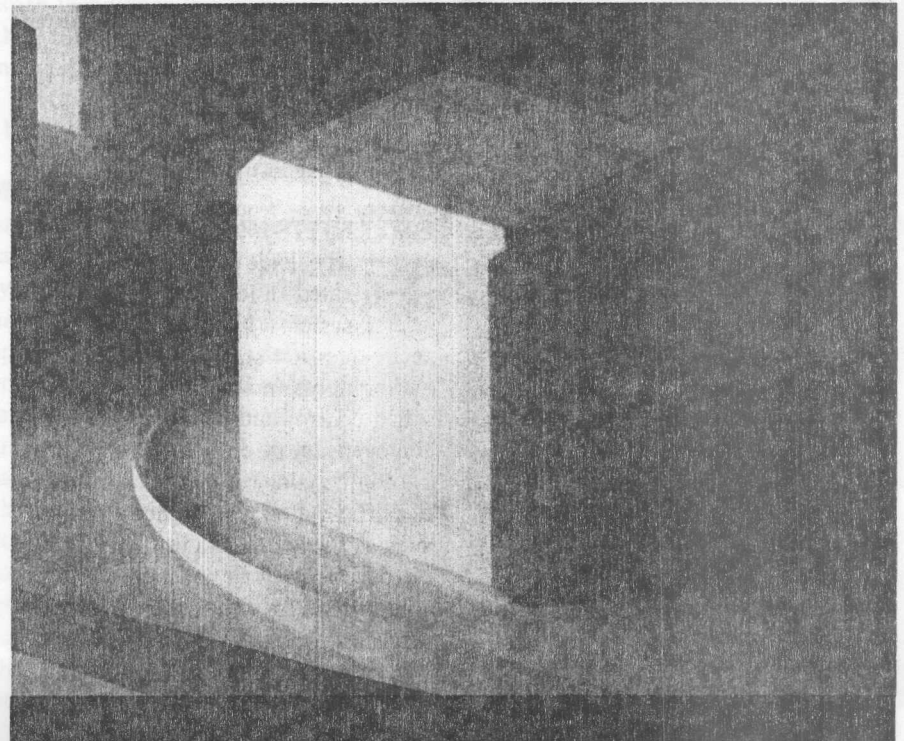
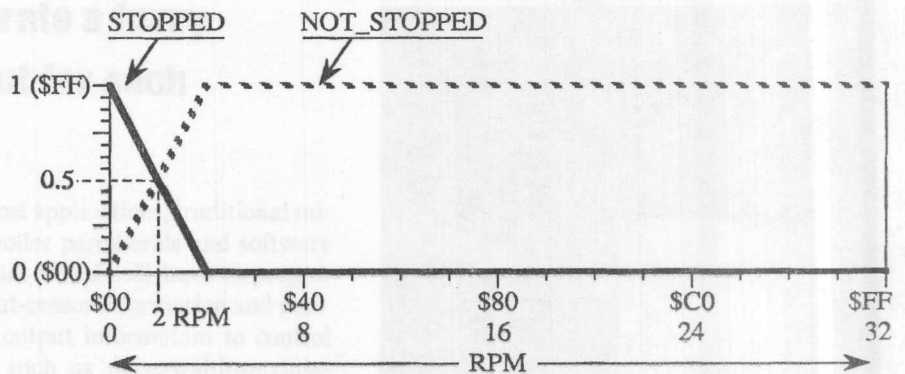
FOLLOWING THE RULES

Fuzzy logic uses rules to define the behavior of a system. A typical fuzzy rule for an automatic watering system might be, "If TEMPERATURE is VERY HOT and DAYS SINCE RAIN is LONG then WATERING TIME is INCREASED GREATLY."

This rule is said to have two antecedents ("if" parts) and one consequent ("then" part). For the software-inference processor discussed later in this article, this rule would be stored as three bytes. The entire sprinkler controller might be defined by about a dozen rules with an average of three antecedents and two consequents—or about 60 bytes for all of the rules.

In a dedicated fuzzy processor, all antecedents are processed in parallel to

Figure 1
Membership functions for STOPPED and NOT _ STOPPED.



Listing 1

A general-purpose fuzzy inference processor.

```

1      * Fuzzy Logic Inference Engine
2      *
3      *** Data structures and variables
4      *
5 0000      ORG      $0000      Beginning of HC11 RAM
6 0000      CURRENT__INS RMB      8      Storage for 8 8-bit inputs
7 0008      FUZ__OUTS  RMB      32      Storage for fuzzy outputs
8 0028      COG__OUTS  RMB      4      Defuzzified outputs
9 002C      LOWEST__IF  RMB      1      Holds min grade of IF parts
10 002D      SUM__OF__FUZ RMB      2      11-bit sum of fuzzy outs
11 002F      SUM__OF__PRODRMB      3      19-bit sum of products
12 0032      COGDEX      RMB      1      Current out # for COG loop 0->4
13 0033      SUMDEX      RMB      1      Index for sum loop 8->0
14
15 B600      ORG      $B600      Beginning of HC11 EEPROM
16 B600 B610      IN__MF__PTRS FDB      IN0MF      Addr of MF data for input 0
17 B602 B618      FDB      IN1MF      Addr of MF data for input 1
18 B604 B61C      FDB      IN2MF      Addr of MF data for input 2
19 B606 B630      FDB      IN3MF      Addr of MF data for input 3
20 B608 B644      FDB      IN4MF      Addr of MF data for input 4
21 B60A B644      FDB      IN5MF      Addr of MF data for input 5
22 B60C B644      FDB      IN6MF      Addr of MF data for input 6
23 B60E B644      FDB      IN7MF      Addr of MF data for input 7
24
25      *
26      * Input membership functions are defined by four 8-bit values per
27      * input label. Up to 8 labels per input so max size of MF data
28      * structure is 8*8*4=256 bytes. Unused labels take no space.
29      * Membership functions are trapezoids with the base greater than or
30      * equal to the top. Values are entered into this program as the
31      * X coordinates of 4 points but are stored as 2 points and 2 slopes.
32      *
33      ***** MACRO Definition for input membership functions
34      *
35      INMF      MACRO      For input membership functions
36      FCB      :0      First inflection point
37      IF :1-:0      Check for divide by zero
38      FCB      ($FF+((:1-:0)/2))/(:1-:0) If not, calc slope
39      ELSE
40      FCB      $00      Indicates vertical slope
41      ENDIF
42      IF :3-:2      Third inflection point
43      FCB      ($FF+((:3-:2)/2))/(:3-:2) If not, calc slope
44      ELSE
45      FCB      $00      Indicates vertical slope

```

Fuzzy Micros

at (415) 905-2689, or the Motorola FREEware BBS at (512) 891-3733. With this program in place, the application programmer develops the membership functions and rules needed to solve the application problem. Software development tools for fuzzy logic are not absolutely required but can simplify the job of developing the membership function and rule databases.

Programmers that develop embedded-control applications are already familiar with preprocessing inputs and post-processing outputs. Preprocessing inputs would involve tasks such as converting tachometer pulses into a value representing the speed of rotation. Scaling may be required to match the input ranges defined for the input-membership functions. Once inputs are processed, the fuzzy processor is called. Post-processing outputs involves converting the defuzzified outputs into control actions such as changing the speed of a motor or applying a braking force.

The fuzzy processor (whether software or hardware) uses linguistic rules to decide what control action to take in response to a given set of input values. Membership functions provide quantified definitions of linguistic terms such as STOPPED, NOT_STOPPED, VERY_HOT, LONG, and so on. Each antecedent is a propositional pairing of an input to a linguistic label that is defined in the range of that input. For example, the antecedent "If TEMPERATURE is VERY_HOT" seeks to determine the degree to which the present value of TEMPERATURE matches the membership function for VERY_HOT. Each antecedent can be evaluated by finding the intersection of the present input value and the membership function of the proposed linguistic label. The fuzzy min operation is used to combine the grades of all antecedents in a rule to give a grade or truth value to the whole rule. The grade for the rule is then applied to the consequents of the rule. Each conse-

48	*				
49	*****				
50 B610	INPUT_MFS	EQU	*	Input membership functions	
52 B610	INOMF	EQU	*	(0) ROTATION	
53 B610 00 00 00 08		INMF	0,0,0,8*4	(0) STOPPED	
66 B614 00 08 FF 00		INMF	0,8*4,\$FF,\$FF	(1) NOT_STOPPED	
79					
80 B618	IN1MF	EQU	*	(1) EXAMPLE1	
81 B618 20 08 60 04		INMF	\$20,\$40,\$60,\$A0	(0) TEST1	
94					
95 B61C	IN2MF	EQU	*	(2) TEMPERATURE	
96 B61C 00 00 50 0D		INMF	2*0,2*0,2*40,2*50	(0) COLD	
109 B620 50 0D 78 0D		INMF	2*40,2*50,2*60,2*70	(1) COOL	
122 B624 78 0D A0 0D		INMF	2*60,2*70,2*80,2*90	(2) WARM	
135 B628 A0 0D BE 09		INMF	2*80,2*90,2*95,2*110	(3) HOT	
148 B62C B4 06 FF 00		INMF	2*90,2*110,\$FF,\$FF	(4) VERY_HOT	
161					
162 B630	IN3MF	EQU	*	(3) DAYS_SINCE_RAIN	
163 B630 00 00 00 20		INMF	\$00,\$00,\$00,\$08	(0) NONE	
176 B634 00 10 10 20		INMF	\$00,\$10,\$10,\$18	(1) SHORT	
189 B638 10 10 28 0B		INMF	\$10,\$20,\$28,\$40	(2) MEDIUM	
202 B63C 30 10 50 10		INMF	\$30,\$40,\$50,\$60	(3) LONG	
215 B640 50 08 FF 00		INMF	\$50,\$70,\$FF,\$FF	(4) VERY_LONG	
228					
229 B644	IN4MF	EQU	*	Not used	
230 B644	IN5MF	EQU	*	Not used	
231 B644	IN6MF	EQU	*	Not used	
232 B644	IN7MF	EQU	*	Not used	
233					
234 B644	SGLTN_POS	EQU	*	Output singleton positions	
235 B644	OUTOMF	EQU	*	(0) WATERING_TIME	
236 B644 00		FCB	\$00	(0) CUT_OFF	
237 B645 10		FCB	\$10	(1) DECREASE_GREATLY	
238 B646 40		FCB	\$40	(2) DECREASE	
239 B647 80		FCB	\$80	(3) NORMAL	
240 B648 B0		FCB	\$B0	(4) INCREASE	
241 B649 F0		FCB	\$F0	(5) INCREASE_GREATLY	
242 B64A 00 00		FCB	0,0	Unused; Fill 8 per output	
243 B64C 00 00 00 00	OUT1MF	FCB	0,0,0,0,0,0,0,0	Not used	
00 00 00 00					
244 B654 00 00 00 00	OUT2MF	FCB	0,0,0,0,0,0,0,0	Not used	
00 00 00 00					
245 B65C 00 00 00 00	OUT3MF	FCB	0,0,0,0,0,0,0,0	Not used	
00 00 00 00					
246	*				
247	* Each If part is of the form 00AA AXXX where AAA is a label (0-7)				
248	* and XXX is an input (0-7). If parts are connected by ANDs. A rule				
249	* may have any number of if parts (usually 2-8). Then parts are of				
250	* the form 100Y YCCC where the MSB set indicates a then part, YY is				
251	* the output number (0-3), and CCC is the output label (singleton				
252	* 0-7). A rule may have any number of then parts (usually 1 or 2).				
253	* A \$FF indicates the end of a series of rules (number not limited).				
254	*				
255	*	FCB	INx + 8*LABa	If input XXX is label AAA	
256	*	FCB	\$80+8*OUTy+LABc	Then output YY is label CCC	
257					

Creating Fuzzy Micros

quent is a pairing of a system output with a linguistic label that is defined for that output. For example, "Then WATERING_TIME is INCREASED_GREATLY."

For each system output, the programmer identifies membership functions for several conditions of that output. In the software fuzzy processor shown in Listing 1, the output-membership functions are singletons, which are the simplest kind of membership function. Each output singleton is given a linguistic label. Each linguistic label for each output gives rise to a fuzzy output. Several rules may use the same fuzzy output in a consequent. When this situation occurs, the rule with the highest grade (the rule that is most true) determines the value of that fuzzy output.

When all rules have been processed, fuzzy outputs must be combined into a single composite action for each system output. A method called center-of-gravity (COG) defuzzification is used in the software program shown in Listing 1. COG defuzzification takes the weighted average of all fuzzy outputs for each system output. Here is the point in the process where seemingly contradictory outputs can be resolved into a consistent output action. Even a rule that is only slightly true contributes to the output action.

INPUT MEMBERSHIP FUNCTIONS

Membership functions provide the quantitative definition for the meaning of linguistic input labels such as VERY_HOT. Figure 3 shows several alternative shapes for membership functions. The trapezoidal shape is used most often in microprocessor-based applications because it does not require complicated mathematical operations that are difficult or time consuming in a low-cost microcontroller. The bell-shaped function is probably more representative of natural phenomenon but the added mathematical com-

Listing 1 (continued)

```

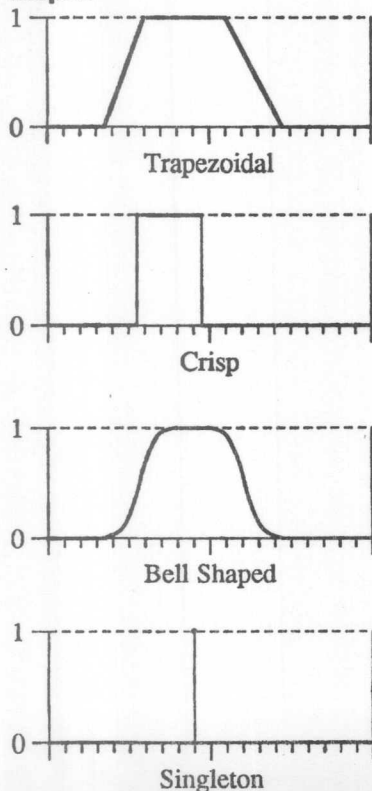
258 B664      RULE_START EQU      *          Start of first rule
259          * Example rule: If TEMPERATURE is VERY_HOT and DAYS_SINCE_RAIN is LONG
260          * Then WATERING_TIME is INCREASE_GREATLY
261 B664 221B85      RULE_1      FCB      2+(8*4),3+(8*3),80+(8*0)+5
262 B667 FF          END_OF_RULECB      $FF
263
264          ***** Fuzzy Inference Engine Starts Here
265          *
266 B668 CE0008      [ 3 ] INFER_TOP      LDX      #FUZ_OUTS      Point at first fuzzy output
267 B66B 8620      [ 2 ]                LDAA      #32          32 fuzzy outputs
268 B66D 6F00      [ 6 ] CLR_OUTS      CLR      0,X          Clear a fuzzy output
269 B66F 08        [ 3 ]                INX                Point at next
270 B670 4A        [ 2 ]                DECA              Loop index
271 B671 26FA      [ 3 ]                BNE      CLR_OUTS      Continue till all fuzzy outs 0
272 B673 18CEB664 [ 4 ]                LDY      #RULE_START      Point to start of 1st rule
273 B677 86FF      [ 2 ] RULE_TOP      LDAA      #$FF          Begin processing rule string
274 B679 972C      [ 3 ]                STAA      LOWEST_IF      Will hold grade of min if part
275 B67B 18E600    [ 5 ] IF_LOOP      LDAB      0,Y          Get rule byte 00AA AXXX; If X is A
276 B67E 2B63      [ 3 ]                BMI      THEN_LOOP      If MSB=1, exit to then loop
277 B680 CE0000    [ 3 ]                LDX      #CURRENT_INS      Point at current input data area
278 B683 C407      [ 2 ]                ANDB      #$07          Save only input number
279 B685 37        [ 3 ]                PSHB              Will need input # again
280 B686 3A        [ 3 ]                ABX                Point to specific input data
281 B687 A600      [ 4 ]                LDAA      0,X          Get current input data
282 B689 CEB600    [ 3 ]                LDX      #IN_MF_PTRS      Point at offsets for input MFs
283 B68C 33        [ 4 ]                PULB              Recover input number 0000 OXXX
284 B68D 3A        [ 3 ]                ABX                Point at pointer for this input #
285 B68E EE00      [ 5 ]                LDX      0,X          Get pointer into MF data area
286 B690 18E600    [ 5 ]                LDAB      0,Y          Get rule if part 00AA AXXX
287 B693 C438      [ 2 ]                ANDB      #$38          00AA A000 is 8 times AAA
288 B695 54        [ 2 ]                LSRB              00AA A000 4 times AAA
289 B696 3A        [ 3 ]                ABX                X points at MF points & slopes
290 B697 A100      [ 4 ]                CMPA      0,X          Compare input data to MF pt1
291 B699 2403      [ 3 ]                BHS      NOT_SEGO      Branch if not segment zero
292 B69B 5F        [ 2 ]                CLRB              In seg 0 grade is zero
293 B69C 2025      [ 3 ]                BRA      HAVE_GRADE      Have grade of membership
294 B69E A102      [ 4 ] NOT_SEGO      CMPA      2,X          Compare input data to MF pt2
295 B6A0 2211      [ 3 ]                BHI      IS_SEG2      Branch if segment two
296 B6A2 E601      [ 4 ]                LDAB      1,X          Slope1 -> B
297 B6A4 2709      [ 3 ]                BEQ      JAM_FF          If vert slope, jam $FF
298 B6A6 A000      [ 4 ]                SUBA      0,X          Input value - pt1 -> A
299 B6A8 3D        [10]                MUL                Grade in B if D < $100
300 B6A9 1A830100 [ 5 ]                CPD      #$100          Check for overflow
301 B6AD 2514      [ 3 ]                BLO      HAVE_GRADE      If < $100 grade OK in B
302 B6AF C6FF      [ 2 ] JAM_FF      LDAB      #$FF          Else limit B to $FF
303 B6B1 2010      [ 3 ]                BRA      HAVE_GRADE      Have grade of membership
304 B6B3 E603      [ 4 ] IS_SEG2      LDAB      3,X          Slope2 -> B
305 B6B5 A002      [ 4 ]                SUBA      2,X          Input value - pt2 -> A
306 B6B7 3D        [10]                MUL                Grade in B if D < $100
307 B6B8 1A830100 [ 5 ]                CPD      #$100          Check for overflow
308 B6BC 2502      [ 3 ]                BLO      B_OK          If < $100 value in B OK
309 B6BE C6FF      [ 2 ]                LDAB      #$FF          Else limit B to $FF
310 B6C0 86FF      [ 2 ] B_OK      LDAA      #$FF          Grade should be $FF - (B)
311 B6C2 10        [ 2 ]                SBA                Grade of membership in B
312 B6C3 D12C      [ 3 ] HAVE_GRADE      CMPB      LOWEST_IF      Is grade lowest so far ?
313 B6C5 2418      [ 3 ]                BHS      NOT_LOWR      Branch if not lower
314 B6C7 D72C      [ 3 ]                STAB      LOWEST_IF      If lower, replace lowest if
315 B6C9 2614      [ 3 ]                BNE      NOT_LOWR      Skip ahead if not zero

```

The fuzzy singleton is a bar of zero width at a specific value in the universe of discourse.

plexity is not justified. The crisp shape gets its name because the boundaries between membership and non-membership are sudden and crisp. Crisp membership functions are consistent with Aristotelian logic. The crisp shape is just a special restrictive case of a trapezoidal membership function, so it follows that nothing will prevent crisp data from being processed by a fuzzy-logic processor. The fuzzy singleton is a bar of zero width at a specific value in the universe of discourse. Singletons are used

Figure 3
Alternate membership-function shapes.



Listing 1 (continued)

316 B6CB 1808	[4]	FIND__THEN	INY		Adv rule pointer to then part
317 B6CD 18E600	[5]	LDAB	0,Y		Get next rule byte
318 B6D0 2AF9	[3]	BPL	FIND__THEN		MSB set means its a then part
319 B6D2 1808	[4]	FIND__IF	INY		Adv rule pointer to if part
320 B6D4 18E600	[5]	LDAB	0,Y		Get next rule byte
321 B6D7 2A9E	[3]	BPL	RULE__TOP		MSB clear means its an if part
322 B6D9 C1FF	[2]	CMPB	#\$FF		\$\$\$ is no more rules marker
323 B6DB 26F5	[3]	BNE	FIND__IF		Continue looking for if or \$\$\$
324 B6DD 2020	[3]	BRA	DEFUZ		When all rules done, go defuzzify
325 B6DF 1808	[4]	NOT__LOWR	INY		Point to next rule byte
326 B6E1 2098	[3]	BRA	IF__LOOP		Continue for all if parts
327 B6E3 CE0008	[3]	THEN__LOOP	LDX	#FUZ__OUTS	Point at fuzzy outputs
328 B6E6 C41F	[2]	ANDB	#\$1F		Save 8 times out # + label #
329 B6E8 3A	[3]	ABX			X points at fuzzy output
330 B6E9 962C	[3]	LDAA	LOWEST__IF		Grade of membership for rule
331 B6EB A100	[4]	CMPA	0,X		Compare to fuzzy output
332 B6ED 2502	[3]	BLO	NOT__HIER		Branch if not higher
333 B6EF A700	[4]	STAA	0,X		Grade is higher so update
334 B6F1 1808	[4]	NOT__HIER	INY		Point to next rule byte
335 B6F3 18E600	[5]	LDAB	0,Y		Get rule byte
336 B6F6 2B03	[3]	BMI	CHK__END		If MSB=0 its a new rule
337 B6F8 7EB677	[3]	JMP	RULE__TOP		Else process next rule byte
338 B6FB C1FF	[2]	CHK__END	CMPB	#\$FF	Check for end of rules flag
339 B6FD 26E4	[3]	BNE	THEN__LOOP		If not \$\$\$, must be a then part
340 B6FF 18CEB644	[4]	DEFUZ	LDY	#SGLTN__POS	Point at 1st output singleton
341 B703 CE0008	[3]	LDX	#FUZ__OUTS		Point at 1st fuzzy output
342 B706 7F0032	[6]	CLR	COGDEX		Loop index will run from 0->4
343 B709 C608	[2]	COG__LOOP	LDAB	#8	8 fuzzy outs per COG output
344 B70B D733	[3]	STAB	SUMDEX		Inner loop runs 8->0
345 B70D CC0000	[3]	LDD	#\$0000		Used for quicker clears
346 B710 DD2D	[4]	STD	SUM__OF__FUZ		Sum of fuzzy outputs
347 B712 DD30	[4]	STD	SUM__OF__PROD+1		Low 16-bits of sum of products
348 B714 92F	[3]	STAA	SUM__OF__PROD		Upper 8-bits
349 B716 E600	[4]	SUM__LOOP	LDAB	0,X	Get a fuzzy output
350 B718 4F	[2]	CLRA			Clear upper 8-bits
351 B719 D32D	[5]	ADDD	SUM__OF__FUZ		Add to sum of fuzzy outputs
352 B71B DD2D	[4]	STD	SUM__OF__FUZ		Update RAM variable
353 B71D A600	[4]	LDAA	0,X		Get fuzzy output again
354 B71F 18E600	[5]	LDAB	0,Y		Get Output singleton position
355 B722 3D	[10]	MUL			Position times weight
356 B723 D330	[5]	ADDD	SUM__OF__PROD+1		Low 16-bits of sum of products
357 B725 DD30	[4]	STD	SUM__OF__PROD+1		Update low 16-bits
358 B727 962F	[3]	LDAA	SUM__OF__PROD		Upper 8-bits
359 B729 8900	[2]	ADCA	#0		Add carry from 16-bit add
360 B72B 92F	[3]	STAA	SUM__OF__PROD		Upper 8-bits of 24-bit sum
361 B72D 1808	[4]	INY			Point at next singleton pos.
362 B72F 08	[3]	INX			Point at next fuzzy output
363 B730 7A0033	[6]	DEC	SUMDEX		Inner loop index
364 B733 26E1	[3]	BNE	SUM__LOOP		For all labels this output
365 B735 3C	[4]	PSHX			Save index for now
366 B736 4F	[2]	CLRA			In case divide by zero
367 B737 DE2D	[4]	LDX	SUM__OF__FUZ		Demominator for divide
368 B739 2718	[3]	BEQ	SAV__OUT		Branch if denominator is 0
369 B73B 7D002F	[6]	TST	SUM__OF__PROD		See if more than 16-bit
370 B73E 2607	[3]	BNE	NUM__BIG		If not zero, # is > 16-bits
371 B740 DC30	[4]	LDD	SUM__OF__PROD+1		Numerator for divide
372 B742 02	[41]	IDIV			Result in low 8-bits of X
373 B743 8F	[3]	XGDX			Result now in B
374 B744 17	[2]	TBA			Move result to A

Fuzzy Micros

for output membership functions.

Before getting too concerned about the exact shape or position of fuzzy membership functions, stop to consider just how well humans are able to solve practical control problems with only their approximate sensors (eyes, ears, hands, and so on). Consider also that different humans approach problems a little differently and still achieve acceptable results. This statement implies that the control methodology must be very robust and able to cope with some variation in sensors, exact membership-function definitions, or both. One method that has been used to demonstrate this robustness is to intentionally remove rules from a rule base and observe the action of the fuzzy control system. Surprisingly, systems continue to operate with more than half of their rules removed. Another way to demonstrate robustness is to purposely change a rule so that the output action requested is exactly the opposite of what it should be. Even this radical attempt to sabotage the control system does not cause some fuzzy control systems to fail, because they have enough other correct rules to compensate for the erroneous one.

Figure 4 shows how input membership functions are specified for the fuzzy-inference program. Four points of inflection are supplied as 8-bit values. The first and fourth points are assumed to define the base of the trapezoid and the second and third points define the top. The Y axis shows the grade of membership as a value between zero and \$FF, which is a slight departure from pure mathematical theory to suit the convenience of the 8-bit microcontroller. In pure fuzzy-set theory, the Y axis should range from 0.0 to 1.0 but, in a real system, we used an 8-bit binary-weighted fraction that ranges from 0.0 (\$00) to 0.99609375 (\$FF).

For input values between the first and second points of inflection, the

377 B749 7D0031	[6]	TST	SUM_OF_PROD+2	Check for rounding error
378 B74C 2A03	[3]	BPL	NO_ROUND	If MSB clear, don't round
379 B74E C30001	[4]	ADDD	#1	Round numerator up 1
380 B751 03	[41] NO_ROUND	FDIV		D/X -> X, use upper 8 of 16
381 B752 8F	[3]	XGDX		Result now in A
382 B753 CE0028	[3] SAV_OUT	LDX	#COG_OUTS	Point to 1st defuz output
383 B756 D632	[3]	LDAB	COGDEX	Current output number
384 B758 3A	[3]	ABX		Point to correct output
385 B759 A700	[4]	STAA	0,X	Update defuzzified output
386 B75B 38	[5]	PULX		Recover index
387 B75C 5C	[2]	INCB		Increment loop index
388 B75D D732	[3]	STAB	COGDEX	Update
389 B75F C104	[2]	CMPB	#4	Done with all four outs?
390 B761 26A6	[3]	BNE	COG_LOOP	If not, continue loop
391				
392				
393 B763				

* Inference engine has completed one pass of all rules.

grade of membership is:

$$\frac{(\text{Input value} - \text{Point 1})}{(\text{Point 2} - \text{Point 1})}$$

Points 1 and 2 are provided at assembly time and the input value is a variable that is determined at run time. To simplify run-time calculations a macro calculates the slope-related values:

$$\frac{1}{(\text{Point 2} - \text{Point 1})}$$

and

$$\frac{1}{(\text{Point 4} - \text{Point 3})}$$

The four points supplied at the time of assembly are translated to a point and a slope starting from the original point 1 and a point and a slope starting from the original point 3. The universe of discourse (X axis) is divided into three segments. The calculation for grade of membership depends on which of these three segments the input value is in. In segment 1, the sloped line segment defined by the point and slope are extended above the \$FF level, as shown by the dashed arrow. During calculations, the grade of membership is limited to a maximum of \$FF. In Figure 4, this limitation corresponds to the portion of the

membership function between \$40 and \$60. A similar limiting action is used in segment 2 to clip the function at \$00.

FUZZY OUTPUTS

Specifying output singletons is analogous to marking settings on a radio volume control. VOLUME would be the the linguistic name for the system output and the marks would correspond to the linguistic labels SOFT, MEDIUM, LOUD, and so on. The position of each fuzzy output in the range of volume would be stored as an 8-bit value in nonvolatile memory (ROM, EPROM, or EEPROM). In Listing 1, the data (output-singleton positions) is structured so the position of each data point identifies the output and the label that is associated with the data value. An identically structured table in RAM holds the fuzzy outputs that are calculated by the fuzzy-inference processor at run time.

When a pass of fuzzy inference starts, all fuzzy outputs are cleared to zero. As rules are processed, the grade for each rule determines the value that will be stored in the fuzzy outputs for the consequents of that rule. If the grade of the current rule is greater than the value stored in the fuzzy output for a consequent of that rule, the fuzzy output will be replaced by the current rule grade. This algorithm implements the fuzzy maximum operation. In ordinary language, this operation means that if two rules try to affect the same fuzzy

Creating Fuzzy Micros

output, the rule that is most true will govern.

DEFUZZIFICATION BY COG

Each output in a system will have several linguistic labels, each of which corresponds to a fuzzy output value in RAM. The object of defuzzification is to combine the effects of all of the fuzzy outputs associated with a system output into a single output value. In some of the first fuzzy-logic systems, the fuzzy output that was largest was taken as the value for the system output. This method was called MAX defuzzification, but professor Bart Kosko has since determined that this method is mathematically inappropriate. The center-of-gravity (COG) method calculates a weighted average of all fuzzy outputs to determine the value to be used as the system output.

The computational difficulty of COG calculations depends upon the shape of output-membership functions. For the software fuzzy processor in this article, output-membership functions are simple singletons. This software program has eight fuzzy outputs per system output (unused fuzzy outputs have a value of zero). The COG calculation is:

$$\frac{\sum_{i=0}^7 (S_i * F_i)}{\sum_{i=0}^7 F_i}$$

Where S_i are the singleton positions from the table in EEPROM and F_i are the corresponding fuzzy output values from the RAM table. In an 8-bit microcontroller, this method is a little harder to use than it appears because the numerator is a 19-bit value (worst case) and the denominator is 11 bits (worst case). Fortunately, the numerator and

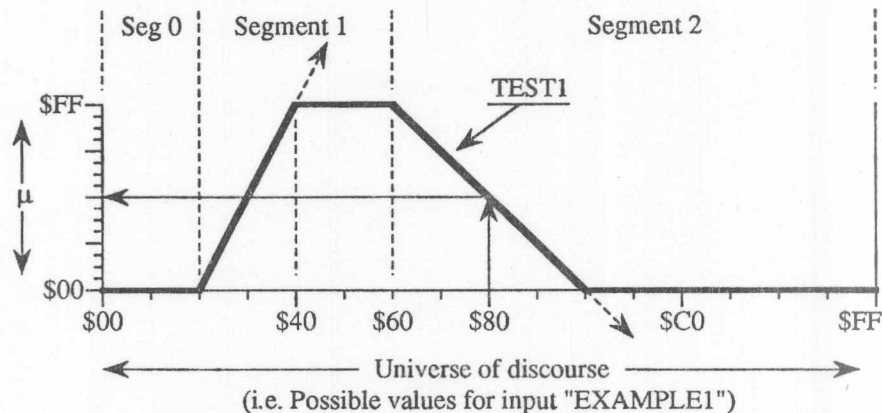
bit value. The upper eight bits of the 24-bit numerator value are first tested to see if the numerator is 16 bits or less. If so, a simple 16-bit-by-16-bit integer-divide instruction (IDIV) can be used to produce the defuzzified result directly.

If the numerator is more than 16 bits, the the M68HC11's fractional-divide instruction (FDIV) can be used to divide the upper 16 bits of the 24-bit

sum of products in the numerator by the 16-bit sum in the denominator. This method is equivalent to dividing the numerator and the result by 256. Using the upper 16 bits of the 24-bit numerator effectively divides it by 256. The result of an FDIV instruction is a binary fraction with the radix point to the left of the MSB. We expected an 8-bit integer value from the original 24-bit-by-

Figure 4

How input-membership functions are specified for the fuzzy-inference program.



How the programmer specifies this membership function.

INMF \$20, \$40, \$60, \$A0

How the MCU works with this membership function.

A macro (INMF) converts four points to two points and two slopes to simplify run time calculations.

point 1, slope 1, point 2, slope 2 = \$20, \$08, \$60, \$04

The calculation of a grade of membership (μ) depends on which segment the input value is in.

In segment 0: $\mu = 0$

In segment 1: $\mu = (\text{Input value} - \text{point 1}) * \text{slope 1}$
limit μ to a maximum value of \$FF

In segment 2: $\mu = \$FF - ((\text{Input value} - \text{point 2}) * \text{slope 2})$
limit μ to a minimum value of \$00

The following calculation shows how the grade of membership is calculated for an input value of \$80.

\$80 is > point 2 (\$60) therefore input is in segment 2

(Input value - point 2) = (\$80 - \$60) = \$20

slope 2 = \$04

((Input value - point 2) * slope 2) = \$20 * \$04 = \$80

$\mu = \$FF - \$80 = \$7F$

Creating Fuzzy Micros

16-bit divide. Moving the radix point eight binary digits to the left is equivalent to dividing the result by 256.

PERFORMANCE ISSUES

The performance of your system is obviously going to depend on how much work you do in the software and how well the processor is suited to the task. For example, the data sheet for an OMRON FP-3000 states that its dedicated-hardware fuzzy processor takes 650 microseconds to process a system of 20 rules with five antecedents and two consequents. A similar system would take up to 8 milliseconds on a 2MHz 68HC11 using the software fuzzy processor in Listing 1. The FP-

The software approach is fast enough for many control applications.

3000 also has 12-bit resolution—vs. 8-bit resolution in this particular software program. While the software approach is slower than the hardware approach, it is fast enough for many practical control applications.

Even simpler microcontrollers such as the M68HC05 can perform software fuzzy inference at the expense of speed. The 68HC05, for example, lacks the IDIV and FDIV instructions and some of the indexing capabilities of the 68HC11, so the defuzzification process would be slower. For small appliances where cost is a major concern, the

slower performance of the fuzzy-logic controls would be unnoticeable.

At the other end of the spectrum, microprocessors with more sophisticated instruction sets and architectures could improve on the software approach. The new 68HC16 can perform 32-bit-by-16-bit divide operations and has multiply and accumulate instructions that should dramatically speed up the defuzzification. Similarly, members of the 68300 family could use the table-look-up-and-interpolate instruction to make the job of finding the grade of membership of inputs much easier and faster.

Jim Sibigtroth is a member of the Motorola technical ladder, the original systems project leader for the M68HC11, and the author of the M68HC11 reference manual. Sibigtroth has also developed printed-circuit-board-level MPU products, including the MEK 6802D5 educational evaluation board. He holds a B.S.E.E. from the University of Illinois.

A STEP BEYOND.

PROMICE takes ROM emulation a step beyond...

An affordable, multi-operational development tool with

ON BOARD INTELLIGENCE

MODULAR DESIGN

SOURCE LEVEL DEBUGGING

FUTURE EXPANDABILITY

The PROMICE enables source level debugging of embedded software by providing communication between the Host and Target systems via the ROM socket. The PROMICE implements ROM monitor based debugging of application code and easily adapts to any target system by simply changing the software required to support the particular target processor



PROMICE... The Next Generation of Firmware Development Tools

For more information, call or write:

Grammar Engine Inc



3314 Morse Road
Columbus, Ohio 43231
TEL 614/471-1113
FAX 614/475-6871

CIRCLE #217 ON READER SERVICE CARD